Application for United States Letters Patent

for

BUS TRANSACTION GENERATOR AND METHOD OF OPERATION

by

William A. Hobbs

Doug Boyce

Kenneth B. Oliver
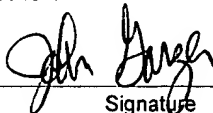
Pierre M. Brasseur

EXPRESS MAIL MAILING LABEL	
NUMBER EM236299499US	
DATE OF DEPOSIT 17 DEC 97	
I hereby certify that this paper or fee is being deposited with the United States Postal Service "EXPRESS MAIL POST OFFICE TO ADDRESSEE" service under 37 C.F.R. 1.10 on the date indicated above and is addressed to: Assistant Commissioner for Patents, Washington D.C. 20231.	

Signature

BUS TRANSAC

BACKGR

1. FIELD OF THE INVENTION

5   The invention relates to a device for generatn.

and more specifically, to a bus transaction/response generator u.

bus cycles under high level language control.

METHOD OF

responses,

cified

2. DESCRIPTION OF RELATED ART

Fig. 1 illustrates a portion of a typical computer system, such as a personal

10   computer (PC), including one or more processors 10 and a chipset 12 coupled together by

means of a processor bus 14. The chipset 12 is coupled to a memory device 15 such as a

dynamic random access memory (DRAM) device and an input/output (I/O) bus 16.

The processor 10 is the "brains" of the computer. The processor 10 includes an

arithmetic logic unit (ALU), which performs arithmetic and logical operations and a

15   control unit, which extracts instructions from memory and decodes and executes them,

calling on the ALU when necessary. A symmetric system is a system that includes

multiple processors, wherein each of the processors is capable of executing any task.

A bus is like a highway on which data travel within a computer. It is simply a

channel over which information flows between two or more devices. A bus normally has

20   access points, or places into which a device can tap to become attached to the bus, and

devices on the bus can send to, and receive information from, other devices. The

processor bus 14 is the bus that the chipset 12 uses to send information to and from the

processor 10. Computer systems also typically include at least one I/O bus 16, such as a peripheral component interconnect (PCI) bus, which is generally used for connecting performance-critical peripherals to the memory 15, chipset 12, and processor 10. For example, video cards, disk storage devices, high-speed networks interfaces generally use

5   a bus of this sort. PCs typically also include additional I/O buses, such as an industry standard architecture (ISA) bus, for slower peripherals such as mice, modems, regular sound cards, low-speed networking, and also for compatibility with older devices.

The system chipset 12 controls this communication, making sure that every device in the system is talking properly to every other device. The chipset 12 is an

10   integrated set of components that perform many of the vital functions of the computer system, including functions that once required several separate chips. Typical functions performed by the chipset 12 include memory controller, which allows the processor 10 to access the computer memory 15; I/O bus bridge, which interfaces the processor bus 14 with other computer buses, such as the PCI or ISA buses; real-time clock (RTC); direct

15   memory access (DMA) controller, which allows system devices to directly access the memory 15 rather than accessing the memory 15 through the processor 10, keyboard controller, mouse controller, etc.

Each transaction initiated on the processor bus 14 goes through three general stages: the arbitration phase, the address phase, and the data phase. For a component, or

20   "agent," on the bus 14 to initiate a transaction on the bus 14, the agent must obtain "ownership" of the bus 14. This happens during the arbitration phase, where the agent initiating the transaction, known as the *requesting agent*, signals that it wants to use the

bus 14. Once the requesting agent acquires bus ownership, it sends an address out on the bus 14 during the address phase. The other agents on the bus 14 receive the address and determine which of them is the target of the transaction -- the *target agent*. Finally, during the data phase, the requesting agent waits for the target agent to provide the

5   requested read data or to accept the write data.

Bus transactions on more complex buses, such as the Intel® Pentium® Pro bus, include additional phases. The Pentium® Pro bus, for example, includes the following transaction phases: arbitration phase, request phase, error phase, snoop phase, response phase, and data phase. The Pentium® Pro bus is divided into groups of signals that

10   generally correspond to the transaction phases, each of which is only used during the respective phase of a transaction. The operation of the Pentium® Pro bus is explained in detail in *Pentium® Pro Processor System Architecture,* by Tom Shanley, MindShare, Inc., chapters 8-17, incorporated herein by reference in its entirety.

When a requesting agent wishes to obtain bus ownership to initiate a bus

15   transaction, it arbitrates for ownership of the request signal group. In a symmetric system (a multiple-processor system wherein each processor is capable of handling any task), multiple processors may simultaneously request ownership of the bus. These processors are referred to as *symmetric agents*, and the process for determining which symmetric agent gets ownership of the bus is known as symmetric arbitration. In comparison,

20   *priority agents* are agents on the bus that arbitrate for bus ownership by asserting a priority signal.

Once the requesting agent has ownership of the request signal group, it drives a transaction request onto the request signal group during the request phase. A Pentium® Pro processor bus transaction request phase is two clocks in duration. Address, transaction type and other transaction information is output in two packets during the

5   request phase; one packet during each of the two clocks of the request phase. The other bus agents receive the transaction request and determine which of them is the target agent.

Moreover, specific bus transactions may take a relatively long time to complete. For instance, the target agent may be busy and therefore, not available to immediately

10   complete the request. In this case, the target agent may choose to defer the transaction request to a later time, in which case the target agent is called a *deferring agent*. Further, data reads and writes may often take a significant amount of time to complete. The Pentium® Pro bus is designed to prevent busy or slow bus agents from tying up the bus for an extended time period. If the target of a data read or write will take a long time to

15   complete the data transfer, it instructs the requesting agent to terminate the transaction, then the targeted agent will initiate a transaction to transfer the data when the read or write is completed.

During the error phase, the bus agents receiving the transaction request inspect the request to determine whether it was corrupted during the transmission by verifying the

20   transaction request's parity bit. Any of the receiving agents that detect a parity error assert the error phase signal. The requesting agent checks the error phase signal at the end of the error phase to see if the request was received without error. If an error was

detected, the remaining transaction phases are canceled and the transaction may be retried. If the receiving agents all receive the request without error, the requesting agent proceeds to the next phase.

5   If the initiated transaction is a memory transaction, agents receiving the transaction request that have an internal cache perform a cache lookup, or "snoop," to determine whether the requested memory line is in any agent's cache. These agents are referred to as *snooping agents*. Snooping agents provide the snoop results via the snoop phase signals. During the snoop phase, the requesting agent and the response agent sample the snoop phase signals. Once the snoop phase is complete, the requesting agent

10   proceeds to the response phase and begins sampling the response phase signals. The response agent notifies the requesting agent how it intends to handle the request via the response phase signals.

Finally, if the transaction involves a data transfer, the requesting agent continues with the data phase. If the bus transaction is a data write transaction, the requesting agent

15   delivers the data to the response agent during the data phase. If it is a data read transaction, the response agent delivers the data to the requesting agent. The bus transaction is complete when the data phase is finished.

The Pentium® Pro bus is designed to allow various phases of several different bus transactions to occur simultaneously to improve bus performance. When a request agent

20   finishes with the signal group for a specific phase, it relinquishes control of that signal group and takes ownership of the signal group for the next phase. For example, once the

H: 226437(4%PX011.DOC)

request agent has issued its transaction request and is ready to check the error phase signals, it no longer requires the use of the request phase signals. Thus, it relinquishes control of the request phase signals and allows another bus agent to take control and initiate a transaction.

5   Components, such as processors and chipsets, coupled to complex buses often contain bugs that cause them to fail during operation. In the failing scenarios, the components frequently respond incorrectly to specific bus transaction sequences, even when the bus transaction sequence is "legal"; in other words, there is nothing in the bus transaction sequence that should have caused the component to fail. The process of

10   correcting these bugs requires inducing the failure under controlled conditions to isolate the bug. Part of this process may require repeating the bus transaction sequence corresponding to the failure.

Current practices for validating bus/component interactions in which failures occur may include operating a processor on a bus using standard assembly language to

15   generate bus transactions. Assembly language, however, has little correlation to the types of bus transactions being generated, offers little control over the exact sequence of bus cycles produced, and offers almost no control over the relative timing between back to back bus transaction phases.

Moreover, this problem is compounded when one or more out-of-order processors

20   are bus agents. An in-order processor fetches a set of instructions and executes them in a sequential program order. On the other hand, an out-of-order processor improves

instruction execution performance by executing instructions that are ready for execution before instructions that are not ready, even if the ready instruction is later in the instruction sequence. This prevents the processor from stalling while an instruction is made ready for execution. For example, a non-ready instruction awaiting data from a

5 memory read does not stall the execution of later instructions in the instruction stream that are ready to execute. Thus, with prior art assembly language controlled, processor-generated bus transactions using an out-of-order processor, the processor controls the bus transaction sequences, rather than a technician attempting to generate a specific sequence of bus transactions.

10 A similar problem occurs in the development of components such as chipsets and processors. In the early design stages, the components being developed are simulated in a software environment and are often driven by a high-level bus model known in the art as a bus functional model (BFM). A simulation stimulus software language may then used to exercise the simulated system. While these methods provide a designer a high level of

15 control, observability and repeatability when testing system designs, the simulation environments are extremely slow, typically operating at about 1-10 Hz. Further, there is no corresponding capability for repeating the simulation on actual systems (hardware) during the design stages that provides the desired control, observability and repeatability.

Thus, a need exists for a system that generates precisely controlled, user-specified

20 bus transactions at an acceptable speed.

SUMMARY OF THE INVENTION

In one aspect of the invention, a system for generating transactions on a bus includes at least one instruction memory storing one or more predefined bus stimuli instructions, and at least one phase generator coupled between the bus and the instruction

5   memory for providing signals to the bus in response to the instructions.

In another aspect of the invention, a method for generating transactions on a bus includes receiving instructions representing predefined bus stimuli. The instructions are converted to signals that, when applied to the bus, execute at least one phase of a bus transaction.

10   In yet another aspect of the invention, a method for verifying the operation of at least one bus agent using a bus transaction generator is presented. The bus transaction generator provides bus stimuli in response to a predefined sequence of bus transactions and in response to signals received from the bus. The method includes coupling at least one bus agent to the bus and coupling the bus transaction generator to the bus. A

15   sequence of bus transactions is defined and assembled into an object file representing bus stimuli. The method further includes initializing the bus agent and executing the bus stimuli.

BRIEF DESCRIPTION OF THE DRAWINGS

20   Other objects and advantages of the invention will become apparent upon reading the following detailed description and upon reference to the drawings in which:

Figure 1 is a block diagram of a prior art computer system;

Figure 2 is a block diagram of a bus transaction generator in accordance with an embodiment of the present invention.

Figure 3 is a block diagram of a bus transaction generator in accordance with

5   another embodiment of the invention;

Figure 4 is a block diagram that illustrates yet another embodiment of a bus transaction generator in accordance with the present invention

Figure 5 is a block diagram illustrating a particular embodiment of a bus transaction generator in accordance with the present invention;

10   Figure 6 illustrates a first exemplary implementation of a transaction generator in accordance with an embodiment of the invention; and

Figure 7 illustrates a second exemplary implementation of a transaction generator in accordance with an embodiment of the invention.

While the invention is susceptible to various modifications and alternative forms,

15   specific embodiments thereof have been shown by way of example in the drawings and are herein described in detail. It should be understood, however, that the description herein of specific embodiments is not intended to limit the invention to the particular forms disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the invention as defined

20   by the appended claims.

H: 226437(4%PX011.DOC)

DETAILED DESCRIPTION OF THE INVENTION

Illustrative embodiments of the invention are described below. In the interest of clarity, not all features of an actual implementation are described in this specification. It

5   will of course be appreciated that in the development of any such actual embodiment, numerous implementation-specific decisions must be made to achieve the developers' specific goals, such as compliance with system-related and business-related constraints, which will vary from one implementation to another. Moreover, it will be appreciated that such a development effort might be complex and time-consuming, but would

10   nevertheless be a routine undertaking for those of ordinary skill in the art having the benefit of this disclosure.

Fig. 2 is a block diagram illustrating an embodiment of a transaction generator 20 in accordance with the present invention. The transaction generator 20 includes an instruction memory 22 that stores one or more predefined representations of bus stimuli

15   and at least one phase generator 24 coupled to a bus 26 that provides the bus stimuli in response to the instructions. The transaction generator 20 is coupled to the bus 26 via an appropriate connector, such as a male connector that mates with a processor socket on a motherboard. In one embodiment of the invention, the phase generator 24 further tracks the state of the bus 26 and responds to stimuli received from the bus 26.

20   Fig. 3 illustrates an embodiment of the present invention in which the phase generator includes a digital logic device 25 and a phase engine 28 coupled between the

H: 226437(4¾PX011.DOC)

instruction memory 22 and the bus 26. The logic device 25 includes the logic circuits and associated support circuits to implement the instructions stored in the instruction memory 22. The phase engine 28 functions to control the timing and logic levels of the bus stimuli. The embodiment illustrated in Fig. 3 also includes a response memory 30, which

5   may store information relating to specific bus transaction phases and predefined responses to stimuli received from the bus 26.

Fig. 4 is a block diagram that illustrates yet another embodiment of a transaction generator 20 in accordance with the present invention. The logic device 25 comprises a control portion 40 for controlling the phases of the bus transactions and a data portion 42.

10   The control portion 40 generally arbitrates for bus ownership and executes the request portions of the bus transaction. The data portion 42 implements the data portions of a bus transaction. A data memory 44 is coupled to the data logic device 42 for storing data that is transmitted to or received from other agents on the bus 26. In a similar manner, the phase engines 28 comprise a control phase engine 46 and a data phase engine 48 for

15   controlling the timing of the bus stimuli.

Fig. 5 is a block diagram illustrating a particular embodiment of a transaction generator 20 in accordance with the present invention. The embodiment of Fig. 5 implements the transaction generator 20 with an Intel® Pentium® Pro processor bus. While Fig. 5 illustrates an implementation of the transaction generator 20 with the

20   Pentium® Pro processor bus, the present invention is not limited to any particular type of bus. It would be well within the ordinary skill in the art to apply the present invention to other bus types by one in the art having the benefit of this disclosure. In the exemplary

embodiment of Fig. 5, the phase generator 24 comprises three field programmable gate

arrays (FPGA) 50, 52, 54. Suitable FPGAs, for example, include model 4036XL or

4062XL FPGAs available from Xilinx® Inc., San Jose, CA. The FPGAs 50, 52, 54 may

be programmed using a hardware descriptive language as is known in the art, such as

5   VHDL or Verilog. Alternately, application specific integrated circuits (ASIC) may be

designed, or discrete logic devices may be arranged, to implement the digital logic

functions of the transaction generator 20 by one skilled in the art having the benefit of

this disclosure.

The three FPGAs 50, 52, 54 of the embodiment illustrated in Fig. 5 comprise a

10   flow FPGA 50, a request FPGA 52 and a data FPGA 54. Each of the FPGAs 50, 52, 54

is coupled to the instruction memory 22, which in an embodiment of the invention

comprises a static random access memory (SRAM) device. The data FPGA 54 is further

coupled to the data memory 44, and the flow FPGA 50 and the request FPGA 52 are

coupled to the response memory 30. The data memory 44 and response memory 30 also

15   may comprise SRAM devices.

A plurality of phase engines 28 are coupled to the bus 26 to control the timing of

the bus stimuli as generated by the FPGAs 50, 52, 54. The phase engines 28 in the

specific embodiment illustrated in Fig. 5 generally correspond to the phases of a

Pentium® Pro bus transaction, including an arbitration phase engine 56, a request phase

20   engine 58, a snoop/error phase engine 60 and a data phase engine 48. If the transaction

generator 20 is implemented with another type of bus, the specific phase engines

employed may be different than illustrated in Fig. 5. Additionally, a system phase engine

necessary programming and control is provided by the system with which it is implemented. In other words, the processor(s) 70 coupled to the processor bus 26, along with the associated control and memory devices 72, 74 coupled to the system bus and other system components (not shown) coupled to the I/O bus 76, may provide the

5   programming and control functions of the host computer.

In a specific embodiment of the invention, bus stimuli and other instructions are formatted into instruction words having a predefined length. In an embodiment of the invention, the instruction words are 216 bits long. The instruction words are divided into predefined segments, each segment providing instructions to various portions of the

10   transaction generator 20. For example, an instruction word may be divided into four segments, including a control portion that identifies the type of operation to be conducted, flow FPGA inputs that provide arbitration instructions and request attributes, response FPGA inputs that provide address and transaction type data, and data FPGA inputs that identify data memory addresses for data writes or reads. The composition of the

15   instruction words will depend on the specific bus stimuli to be generated and the specific type of bus with which the transaction generator is implemented.

In one embodiment, such as the embodiment illustrated in Fig. 5, the transaction generator 20 is implemented with a Pentium® Pro processor bus 26, though the transaction generator 20 may be implemented with any type of bus by one in the art

20   having the benefit of this disclosure. Thus, a bus transaction begins by arbitrating for bus ownership. For example, if the transaction generator 20 is coupled to the bus 26 in place of one processor in a multi-processor system, such as the Pentium® Pro system, it would

assert an arbitration signal as a symmetric agent to request ownership of the request phase

signals. If the transaction generator 20 is operating as a priority agent, such as a host/PCI

bridge, it would assert the arbitration signal appropriate for a priority agent. The

arbitration FPGA additionally may provide a request attribute, for example, a LOCK#

5   signal to prevent a priority agent from taking ownership of the bus.

During the request phase of a Pentium® Pro system bus transaction, information

about the transaction is output in two packets, packet A and packet B. During each

packet, information is output on the bus address and request signal groups. Thus, the

transaction generator instruction word request segment allows the user to specify what

10   information is output to the bus 26 on the address and request signal groups in packet A

and packet B. Similarly, the data FPGA portion of the instruction word allows a user to

specify appropriate signals for the data phase of the transaction. The data memory 44

stores the data required (read or write) for the data phase of the transaction.

The response memory 30 stores the predefined responses to signals received from

15   the bus during the error, snoop, and response phases of a bus transaction. For example,

the transaction may be programmed to generate a HITM# signal indicating a snoop hit on

a modified line. If the transaction generator 20 is functioning as a response agent, it may

be programmed to assert a DEFER# signal to indicate that it intends to defer the response

to the transaction until a later time.

20   The system configuration illustrated in Fig. 6 may be used to test or verify various

system components and configurations. Further, the test system may be modified from

that illustrated in Fig. 6 to facilitate testing specific bus agents. For example, the configuration of Fig. 6 may be useful for validating the chipset 72 operations against processor bus stimuli in situations where the processor 70 is under development, prior to processor availability. It may also be used to generate system fault sequences that a

5   processor 70 cannot mimic. To test the chipset 72, for example, a series of processor bus stimuli are defined, then assembled into a binary object file that is provided to the transaction generator 20. The system is initialized and the transaction generator 20 implements the bus stimuli. The logic analyzer 78 may then capture the chipset 72 responses during each bus clock for subsequent analysis.

10   Additionally, the configuration of Fig. 6 is useful for providing controlled, "background" bus traffic when testing system operation. System bugs often occur during a specific sequence of bus transactions. Often, these bugs do not repeat under controlled scenarios with minimal bus traffic. The transaction generator 20 may be used to generate high volumes of complex bus transactions, making bugs more apparent. Interagent

15   transactions may be investigated, and fault sequences may be repeated under controlled conditions. A series of bus transactions may be defined, then programmed to repeat, or loop, for a predetermined number of cycles to stress the bus. Further, the transaction generator may be programmed to intentionally generate errors. The predefined sequence of bus transactions may include "illegal" bus sequences, for example, to determine the

20   ability of the system under test to respond to faults.

Initially, background bus traffic may be generated without data checking to quickly identify catastrophic failures, or "splats," such as a complete system lock-up.

More sophisticated tests, which include data checking, may be conducted. For example, the transaction generator 20 may be programmed to write preselected data to a location in the memory device 74, then the processor 70 is instructed to read the data. Any data discrepancies may then be correlated to the logic analyzer trace file to isolate the source

5   of the problem. Alternatively, the transaction generator 20 may be programmed to compare the write data to the read data, and record any discrepancies for later analysis.

Fig. 7 illustrates an alternate implementation of the transaction generator 20 that may be used to test the processor 70 operations that are unsupported by an existing chipset. At least one processor 70 is coupled to the bus 26, along with the transaction

10   generator 20. The transaction generator 20 is used to emulate protocol and data operations for agents addressed by the processor 70. In a manner similar to the configuration of Fig. 6, fault sequences may be repeated under program control.

The particular embodiments disclosed above are illustrative only, as the invention may be modified and practiced in different but equivalent manners apparent to those

15   skilled in the art having the benefit of the teachings herein. For example, the transaction generator could be configured to operate with virtually any type of bus by one skilled in the art having the benefit of this disclosure. Furthermore, no limitations are intended to the details of construction or design herein shown, other than as described in the claims below. It is therefore evident that the particular embodiments disclosed above may be

20   altered or modified and all such variations are considered within the scope and spirit of the invention. Accordingly, the protection sought herein is as set forth in the claims below.

8.   The system of claim 6 wherein the digital logic device comprises an application specific integrated circuit.

9.   The system of claim 6 wherein the at least one digital logic device includes a control portion for providing bus control signals and a data portion for sending

5   data to the bus.

10.   The system of claim 9 wherein the control portion includes a flow logic device, a request logic device, and a data logic device.

11.   The system of claim 6 wherein the at least one phase engine includes at least one logic level translation device.

10   12.   The system of claim 6 wherein the at least one phase engine comprises a system phase engine, an arbitration phase engine, a request phase engine, a snoop/error phase engine, and a data phase engine.

13.   The system of claim 9 further comprising a data memory coupled to the data portion.

15   14.   The system of claim 9 wherein the data portion further receives data from the bus.

15.   A system for generating transactions on a bus comprising:

an instruction memory storing digital data representing predefined bus stimuli;

a flow logic device responsive to the instruction memory;

H: 226437(4%PX011.DOC)

70 are coupled to the bus 26 in addition to the transaction generator 20, though the transaction generator 20 is capable of interacting with the bus 26 and other bus agents, such as the chipset 72, by itself with no additional processors 70 coupled to the bus 26. The chipset 72 is coupled to a memory device 74 and also provides a bridge to other

5   buses 76 in the system, such as a PCI bus, ISA bus, etc. A logic analyzer 78 may be coupled to the bus 26 to capture the state of the devices coupled to the bus 26 during the bus transactions. The transaction generator 20 is coupled to a host computer system 80, such as a personal computer (PC).

A bus stimulus file 82 that may include a series of desired bus transactions is

10 stored in the host system. The bus stimulus file 82 may be based on a user-defined series of bus transactions, an existing system simulation based on a BFM, a logic analyzer trace file, etc. The bus stimulus file 82 is assembled into a binary object file 84 that is provided to the transaction generator 20 along with set-up information.

The host computer 80 provides a programming and control interface to the

15 transaction generator 20. In addition to providing the binary object file 84 containing the instructions to the instruction memory of the transaction generator 20, the host computer 80 programs the various logic devices in the transaction generator using standard software packages available for this purpose. Further, the host computer 80 may exchange data with the data memory of the transaction generator 80, and provide the

20 proper responses to received bus signals to the transaction generator 80 response memory. Alternatively, the transaction generator 20 may be implemented without requiring a host computer 80. The transaction generator 20 may be configured such that

H: 226437(4%PX011.DOC)